300 Core Java Interview Questions

1) What is Java?

Java is the high-level, object-oriented, robust, secure programming language, platform-independent, high performance, Multithreaded, and portable programming language. It was developed by **James Gosling** in June 1991. It can also be known as the platform as it provides its own JRE and API.

2) List the features of Java Programming language.

There are the following features in Java Programming Language.

- **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.
- **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.
- **Platform Independent:** Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.
- **Secured:** Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.
- **Robust:** Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.
- **Interpreted:** Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.
- **Multithreaded:** We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.
- **Dynamic:** Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

3) What do you understand by Java virtual machine?

Java Virtual Machine is a virtual machine that enables the computer to run the Java program. JVM acts like a run-time engine which calls the main method present in the Java code. JVM is the specification which must be implemented in the computer system. The Java code is compiled by JVM to be a Bytecode which is machine independent and close to the native code.

4) What is the difference between JDK, JRE, and JVM?

JVM

JVM is an acronym for Java Virtual Machine; it is an abstract machine which provides the runtime environment in which Java bytecode can be executed. It is a specification which specifies the working of Java Virtual Machine. Its implementation has been provided by Oracle and other companies. Its implementation is known as JRE.
JVMs are available for many hardware and software platforms (so JVM is platform dependent). It is a runtime instance which is created when we run the Java class. There are three notions of the JVM: specification, implementation, and instance.

JRE

JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

5) How many types of memory areas are allocated by JVM?

Many types:

1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects
3. **Stack:** Java Stack stores **frames**. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

6) What is JIT compiler?

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

7) What gives Java its 'write once and run anywhere' nature?

The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This bytecode is not platform specific and can be executed on any computer.

8) What is classloader?

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**
2. **Extension ClassLoader**
3. **System/Application ClassLoader**

9) Is Empty .java file name a valid source file name?

Yes, Java allows to save our java file by **.java** only, we need to compile it by **javac .java** and run by **java classname** Let's take a simple example:

```
//save by .java only
class A{
public static void main(String args[]){
System.out.println("Hello java");
}
 }
//compile by javac .java
//run by     java A
```

compile it by **javac .java**

run it by **java A**

10) Is delete, next, main, exit or null keyword in java?

No.

11) If I don't provide any arguments on the command line, then what will the value stored in the String array passed into the main() method, empty or NULL?

It is empty, but not null.

12) What if I write static public void instead of public static void?

The program compiles and runs correctly because the order of specifiers doesn't matter in Java.

13) What is the default value of the local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

14) What are the various access specifiers in Java?

In Java, access specifiers are the keywords which are used to define the access scope of the method, class, or a variable. In Java, there are four access specifiers given below.

- **Public** The classes, methods, or variables which are defined as public, can be accessed by any class or method.
- **Protected** Protected can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.
- **Default** Default are accessible within the package only. By default, all the classes, methods, and variables are of default scope.
- **Private** The private class, methods, or variables defined as private can be accessed within the class only.

15) What is the purpose of static methods and variables?

The methods or variables defined as static are shared among all the objects of the class. The static is the part of the class and not of the object. The static variables are stored in the class area, and we do not need to create the object to access such variables. Therefore, static is used in the case, where we need to define variables or methods which are common to all the objects of the class.

For example, In the class simulating the collection of the students in a college, the name of the college is the common attribute to all the students. Therefore, the college name will be defined as **static**.

16) What are the advantages of Packages in Java?

There are various advantages of defining packages in Java.

- Packages avoid the name clashes.
- The Package provides easier access control.
- We can also have the hidden classes that are not visible outside and used by the package.
- It is easier to locate the related classes.

17) What is the output of the following Java program?

```
class Test
{
    public static void main (String args[])
    {
        System.out.println(10 + 20 + "Javatpoint");
    System.out.println("Javatpoint" + 10 + 20);
    }
}
```

The output of the above code will be

```
30Javatpoint
Javatpoint1020
```

Explanation

In the first case, 10 and 20 are treated as numbers and added to be 30. Now, their sum 30 is treated as the string and concatenated with the string **Javatpoint**. Therefore, the output will be **30Javatpoint**.

In the second case, the string Javatpoint is concatenated with 10 to be the string **Javatpoint10** which will then be concatenated with 20 to be **Javatpoint1020**.

18) What is the output of the following Java program?

```
class Test
{
    public static void main (String args[])
    {
        System.out.println(10 * 20 + "Javatpoint");
    System.out.println("Javatpoint" + 10 * 20);
    }
}
```

The output of the above code will be

```
200Javatpoint
Javatpoint200
```

Explanation

In the first case, The numbers 10 and 20 will be multiplied first and then the result 200 is treated as the string and concatenated with the string **Javatpoint** to produce the output **200Javatpoint**.

In the second case, The numbers 10 and 20 will be multiplied first to be 200 because the precedence of the multiplication is higher than addition. The result 200 will be treated as the string and concatenated with the string **Javatpoint**to produce the output as **Javatpoint200**.

19) What is the output of the following Java program?

```
class Test
{
    public static void main (String args[])
    {
        for(int i=0; 0; i++)
        {
            System.out.println("Hello Javatpoint");
        }
    }
}
```

The above code will give the compile-time error because the for loop demands a boolean value in the second part and we are providing an integer value, i.e., 0.

20) What is an object?

The Object is the real-time entity having some state and behavior. In Java, Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the **new** keyword.

21) What will be the initial value of an object reference which is defined as an instance variable?

All object references are initialized to null in Java.

Core Java - OOPs Concepts: Constructor Interview Questions

22) What is the constructor?

The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

23) How many types of constructors are used in Java?

Based on the parameters passed in the constructors, there are two types of constructors in Java.

- **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
- **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.

24) What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

```
class Student3{
```

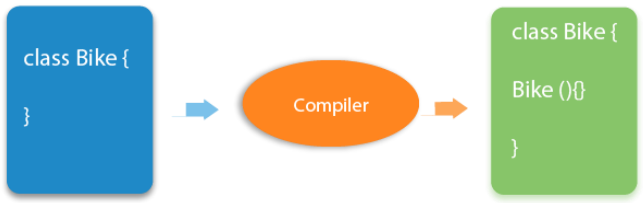
```
int id;
String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

Test it Now

Output:

```
0 null
0 null
```

Explanation: In the above class, you are not creating any constructor, so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.



25) Does constructor return any value?

Ans: yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor).

26) Is constructor inherited?

No, The constructor is not inherited.

27) Can you make a constructor final?

No, the constructor can't be final.

28) Can we overload the constructors?

Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. Consider the following example.

```
class Test
{
    int i;
    public Test(int k)
    {
        i=k;
    }
    public Test(int k, int m)
    {
        System.out.println("Hi I am assigning the value max(k, m) to i");
        if(k>m)
        {
            i=k;
        }
        else
        {
            i=m;
        }
    }
}
public class Main
{
    public static void main (String args[])
    {
        Test test1 = new Test(10);
        Test test2 = new Test(12, 15);
        System.out.println(test1.i);
        System.out.println(test2.i);
    }
}

```

In the above program, The constructor Test is overloaded with another constructor. In the first call to the constructor, The constructor with one argument is called, and i will be initialized with the value 10. However, In the second call to the constructor, The constructor with the 2 arguments is called, and i will be initialized with the value 15.

29) What do you understand by copy constructor in Java?

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object classIn this example, we are going to copy the values of one object into another using javaconstructor.

```java
//Java program to initialize the values from one object to another
class Student6{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n){
    id = i;
    name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

Test it Now

Output:

```
111 Karan
111 Karan
```

30) What are the differences between the constructors and methods?

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of anobject.	A method is used to expose thebehavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.

The Java compiler provides a default constructor ifyou don't have any constructor in a class.	The method is not provided by thecompiler in any case.
The constructor name must be same as the classname.	The method name may or may notbe same as class name.

31) What is the output of the following Java program?

```
public class Test
{
    Test(int a, int b)
    {
        System.out.println("a = "+a+" b = "+b);
    }
    Test(int a, float b)
    {
        System.out.println("a = "+a+" b = "+b);
    }
    public static void main (String args[])
    {
        byte a = 10;
        byte b = 15;
        Test test = new Test(a,b);
    }
}
```

The output of the following program is:

```
a = 10 b = 15
```

Here, the data type of the variables a and b, i.e., byte gets promoted to int, and the first parameterized constructor with the two integer parameters is called.

32) What is the output of the following Java program?

```
class Test
{
    int i;
}
public class Main
{
    public static void main (String args[])
    {
        Test test = new Test();
        System.out.println(test.i);
    }
}
```

The output of the program is 0 because the variable i is initialized to 0 internally. As we know that a default constructor is invoked implicitly if there is no constructor in the class, the variable i is initialized to 0 since there is no constructor in the class.

33) What is the output of the following Java program?

```
class Test
{
    int test_a, test_b;
    Test(int a, int b)
    {
    test_a = a;
    test_b = b;
    }
    public static void main (String args[])
    {
        Test test = new Test();
        System.out.println(test.test_a+" "+test.test_b);
    }
}
```

There is a **compiler error** in the program because there is a call to the default constructor in the main method which is not present in the class. However, there is only one parameterized constructor in the class Test. Therefore, no default constructor is invoked by the constructor implicitly.

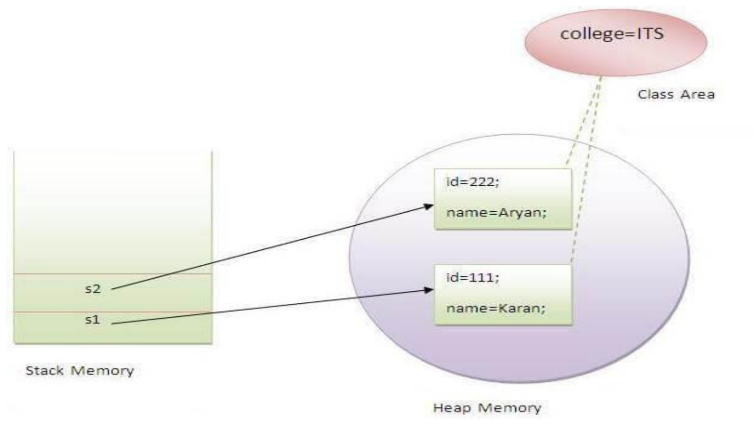
34) What is the static variable?

The static variable is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc. Static variable gets memory only once in the class area at the time of class loading. Using a static variable makes your program more memory efficient (it saves memory). Static variable belongs to the class rather than the object.

```
//Program of static variable

 class Student8{
   int rollno;
   String name;
   static String college ="ITS";

   Student8(int r,String n){
   rollno = r;
   name = n;
   }
void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){
Student8 s1 = new Student8(111,"Karan");
Student8 s2 = new Student8(222,"Aryan");

s1.display();
s2.display();
}
}
```

Test it Now

```
Output:111 Karan ITS
       222 Aryan ITS
```



35) What is the static method?

- A static method belongs to the class rather than the object.
- There is no need to create the object to call the static methods.
- A static method can access and change the value of the static variable.

36) What are the restrictions that are applied to the Java static methods?

Two main restrictions are applied to the static methods.

- The static method can not use non-static data member or call the non-static method directly.
- this and super cannot be used in static context as they are non-static.

37) Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation.

38) Can we override the static methods?

No, we can't override static methods.

39) What is the static block?

Static block is used to initialize the static data member. It is executed before the main method, at the time of classloading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
  System.out.println("Hello main");
  }
}
```

Test it Now

```
Output: static block is invoked
       Hello main
```

40) Can we execute a program without main() method?

Ans) Yes, one of the ways to execute the program without the main method is using static block.

41) What if the static modifier is removed from the signature of the main method?

Program compiles. However, at runtime, It throws an error "NoSuchMethodError."

42) What is the difference between static (class) method and instance method?

static or class method	instance method
1)A method that is declared as static is known as the staticmethod.	A method that is not declared as static is known as the instance method.
2)We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.

4)For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

43) Can we make constructors static?

As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

44) Can we make the abstract methods static in Java?

In Java, if we make the abstract methods static, It will become the part of the class, and we can directly call it which is unnecessary. Calling an undefined method is completely useless therefore it is not allowed.

45) Can we declare the static variables and methods in an abstract class?

Yes, we can declare static variables and methods in an abstract method. As we know that there is no requirement to make the object to access the static context, therefore, we can access the static context declared inside the abstract class by using the name of the abstract class. Consider the following example.

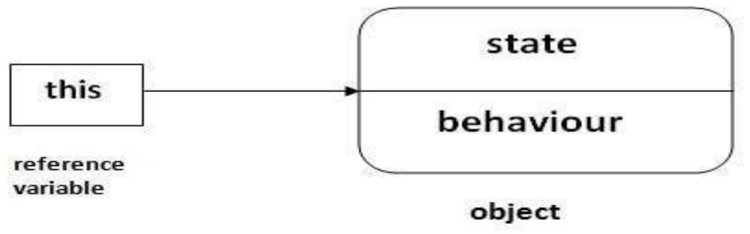
```
abstract class Test
{
    static int i = 102;
    static void TestMethod()
    {
        System.out.println("hi !! I am good !!");
    }
 }
 public class TestClass extends Test
{
    public static void main (String args[])
    {
        Test.TestMethod();
        System.out.println("i = "+Test.i);
    }
}
```

Output

```
hi !! I am good !!
i = 102
```

46) What is **this** keyword in java?

The **this** keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors. It can also be returned from the method as the current class instance.



47) What are the main uses of this keyword?

There are the following uses of **this** keyword.

- **this** can be used to refer to the current class instance variable.
- **this** can be used to invoke current class method (implicitly)
- **this()** can be used to invoke the current class constructor.
- **this** can be passed as an argument in the method call.
-
- **this** can be passed as an argument in the constructor call.
- **this** can be used to return the current class instance from the method.

48) Can we assign the reference to **this** variable?

No, this cannot be assigned to any value because it always points to the current class object and this is the final reference in Java. However, if we try to do so, the compiler error will be shown. Consider the following example.

```
public class Test
{
    public Test()
    {
        this = null;
    System.out.println("Test class constructor called");
    }
    public static void main (String args[])
```

```java
    {
        Test t = new Test();
    }
}
```

Output

```
Test.java:5: error: cannot assign a value to final variable this
        this = null;
        ^
1 error
```

49) Can this keyword be used to refer static members?

Yes, It is possible to use this keyword to refer static members because this is just a reference variable which refers to the current class object. However, as we know that, it is unnecessary to access static variables through objects, therefore, it is not the best practice to use this to refer static members. Consider the following example.

```java
public class Test
{
    static int i = 10;
    public Test ()
    {
        System.out.println(this.i);
    }
    public static void main (String args[])
    {
        Test t = new Test();
    }
}
```

Output

```
10
```

50) How can constructor chaining be done using this keyword?

Constructor chaining enables us to call one constructor from another constructor of the class with respect to the current class object. We can use this keyword to perform constructor chaining within the same class. Consider the following example which illustrates how can we use this keyword to achieve constructor chaining.

```java
public class Employee
{
    int id,age;
    String name, address;
    public Employee (int age)
    {
        this.age = age;
    }
    public Employee(int id, int age)
    {
        this(age);
        this.id = id;
    }
    public Employee(int id, int age, String name, String address)
    {
        this(id, age);
        this.name = name;
        this.address = address;
    }
    public static void main (String args[])
    {
        Employee emp = new Employee(105, 22, "Vikas", "Delhi");
        System.out.println("ID: "+emp.id+" Name:"+emp.name+" age:"+emp.age+" address
: "+emp.address);
    }

}
```

Output

```
ID: 105 Name:Vikas age:22 address: Delhi
```

51) What are the advantages of passing this into a method instead of the current class object itself?

As we know, that this refers to the current class object, therefore, it must be similar to the current class object. However, there can be two main advantages of passing this into a method instead of the current class object.

- this is a final variable. Therefore, this cannot be assigned to any new value whereas the current class object might not be final and can be changed.
- this can be used in the synchronized block.

52) What is the Inheritance?

Inheritance is a mechanism by which one object acquires all the properties and behavior of another object of another class. It is used for Code Reusability and Method Overriding. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

There are five types of inheritance in Java.

- Single-level inheritance
- Multi-level inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Multiple inheritance is not supported in Java through class.

More Details.

53) Why is Inheritance used in Java?

There are various advantages of using inheritance in Java that is given below.

- Inheritance provides code reusability. The derived class does not need to redefine the method of base class unless it needs to provide the specific implementation of the method.
- Runtime polymorphism cannot be achieved without using inheritance.
- We can simulate the inheritance of classes with the real-time objects which makes OOPs more realistic.
- Inheritance provides data hiding. The base class can hide some data from the derived class by making it private.
- Method overriding cannot be achieved without inheritance. By method overriding, we can give a specific implementation of some basic method contained by the base class.

54) Which class is the superclass for all the classes?

The object class is the superclass of all other classes in Java.

55) Why is multiple inheritance not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since the compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have the same method or different, there will be a compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
```

```
class C extends A,B{//suppose if it were

 Public Static void main(String args[]){
  C obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

Test it Now

```
Compile Time Error
```

56) What is aggregation?

Aggregation can be defined as the relationship between two classes where the aggregate class contains a reference to the class it owns. Aggregation is best described as a **has-a** relationship. For example, The aggregate class Employee having various fields such as age, name, and salary also contains an object of Address class having various fields such as Address-Line 1, City, State, and pin-code. In other words, we can say that Employee (class) has an object of Address class. Consider the following example.

Address.java

```
public class Address {
String city,state,country;

public Address(String city, String state, String country) {
    this.city = city;
    this.state = state;
    this.country = country;
}

}
```

Employee.java

```
public class Emp {
int id;
String name;
Address address;

public Emp(int id, String name,Address address) {
    this.id = id;
    this.name = name;
    this.address=address;
}

void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}

public static void main(String[] args) {
Address address1=new Address("gzb","UP","india");
Address address2=new Address("gno","UP","india");

Emp e=new Emp(111,"varun",address1);
Emp e2=new Emp(112,"arun",address2);

e.display();
e2.display();

}
}
```

Output

```
111 varun
gzb UP india
112 arun
gno UP india
```

57) What is composition?

Holding the reference of a class within some other class is known as composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. In other words, we can say that composition is the particular case of aggregation which represents a stronger relationship between two objects. Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

58) What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

59) Why does Java not support pointers?

The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe(unsecured) and complex to understand.

60) What is super in java?

The **super** keyword in Java is a reference variable that is used to refer to the immediate parent class object. Whenever you create the instance of the subclass, an instance of the parent class is created implicitly which is referred by super reference variable. The super() is called in the class constructor implicitly by the compiler if there is no super or this.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}
}
```

Test it Now

Output:

```
animal is created
dog is created
```

More Details.

61) How can constructor chaining be done by using the super keyword?

```
class Person
{
    String name,address;
    int age;
    public Person(int age, String name, String address)
    {
        this.age = age;
        this.name = name;
        this.address = address;
    }
}
class Employee extends Person
{
    float salary;
    public Employee(int age, String name, String address, float salary)
    {
        super(age,name,address);
        this.salary = salary;
    }
}
public class Test
```

```
{
    public static void main (String args[])
    {
        Employee e = new Employee(22, "Mukesh", "Delhi", 90000);
        System.out.println("Name: "+e.name+" Salary: "+e.salary+" Age: "+e.age+" Address: "+e.address);
    }
}
```

Output

```
Name: Mukesh Salary: 90000.0 Age: 22 Address: Delhi
```

62) What are the main uses of the super keyword?

There are the following uses of super keyword.

- super can be used to refer to the immediate parent class instance variable.
- super can be used to invoke the immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

63) What are the differences between this and super keyword?

There are the following differences between this and super keyword.

- The super keyword always points to the parent class contexts whereas this keyword always points to the current class context.
- The super keyword is primarily used for initializing the base class variables within the derived class constructor whereas this keyword primarily used to differentiate between local and instance variables when passed in the class constructor.
- The super and this must be the first statement inside constructor otherwise the compiler will throw an error.

64) What is the output of the following Java program?

```
class Person
{
    public Person()
    {
        System.out.println("Person class constructor called");
    }
}
public class Employee extends Person
{
    public Employee()
    {
        System.out.println("Employee class constructor called");
```

```
  }
  public static void main (String args[])
  {
      Employee e = new Employee();
  }
}
```

Output

```
Person class constructor called
Employee class constructor called
```

Explanation

The super() is implicitly invoked by the compiler if no super() or this() is included explicitly within the derived class constructor. Therefore, in this case, The Person class constructor is called first and then the Employee class constructor is called.

65) Can you use this() and super() both in a constructor?

No, because this() and super() must be the first statement in the class constructor.

Example:

```
public class Test{
    Test()
     {
         super();
         this();
     System.out.println("Test class object is created");
     }
     public static void main(String []args){
     Test t = new Test();
     }
}
```

Output:

```
Test.java:5: error: call to this must be first statement in constructor
```

66) What is method overloading?

Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature. We can achieve method overloading in two ways.

- Changing the number of arguments
- Changing the return type

Method overloading increases the readability of the program. Method overloading is performed to figure out the program quickly.

67) Why is method overloading not possible by changing the return type in java?

In Java, method overloading is not possible by changing the return type of the program due to avoid the ambiguity.

```java
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Test it Now

Output:

```
Compile Time Error: method add(int, int) is already defined in class Adder
```

More Details.

68) Can we overload the methods by making them static?

No, We cannot overload the methods by just applying the static keyword to them(number of parameters and types are the same). Consider the following example.

```java
public class Animal
{
    void consume(int a)
    {
        System.out.println(a+" consumed!!");
    }
    static void consume(int a)
    {
        System.out.println("consumed static "+a);
    }
    public static void main (String args[])
    {
        Animal a = new Animal();
        a.consume(10);
        Animal.consume(20);
16.    }
17. }
```

Output

```
Animal.java:7: error: method consume(int) is already defined in class Animal
    static void consume(int a)
                ^
Animal.java:15:  error:  non-static  method  consume(int)  cannot  be  referenced
from a static context
        Animal.consume(20);
              ^
2 errors
```

69) Can we overload the main() method?

Yes, we can have any number of main methods in a Java program by using method overloading.

78) What is method overriding:

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods.

Rules for Method overriding

- The method must have the same name as in the parent class.
- The method must have the same signature as in the parent class.
- Two classes must have an IS-A relationship between them.

79) Can we override the static method?

No, you can't override the static method because they are the part of the class, not the object.

---

80) Why can we not override static method?

It is because the static method is the part of the class, and it is bound with class whereas instance method is bound with the object, and static gets memory in class area, and instance gets memory in a heap.

---

81) Can we override the overloaded method?

Yes.

82) Difference between method Overloading and Overriding.

Method Overloading	Method Overriding
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes that have IS-A relationship between them.
3) In this case, the parameters must be different.	In this case, the parameters must be the same.

83) Can we override the private methods?

No, we cannot override the private methods because the scope of private methods is limited to the class and we cannot access them outside of the class.

84) Can we change the scope of the overridden method in the subclass?

Yes, we can change the scope of the overridden method in the subclass. However, we must notice that we cannot decrease the accessibility of the method. The following point must be taken care of while changing the accessibility of the method.

- The private can be changed to protected, public, or default.
- The protected can be changed to public or default.
- The default can be changed to public.
- The public will always remain public.

85) Can we modify the throws clause of the superclass method while overriding it in the subclass?

Yes, we can modify the throws clause of the superclass method while overriding it in the subclass. However, there are some rules which are to be followed while overriding in case of exception handling.

- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception, but it can declare the unchecked exception.
- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

86) What is the output of the following Java program?

```java
class Base
{
    void method(int a)
    {
        System.out.println("Base class method called with integer a = "+a);
    }

    void method(double d)
    {
        System.out.println("Base class method called with double d ="+d);
    }
}

class Derived extends Base
{
    @Override
    void method(double d)
    {
        System.out.println("Derived class method called with double d ="+d);
    }
}

public class Main
{
    public static void main(String[] args)
    {
        new Derived().method(10);
    }
}
```

Output

```
Base class method called with integer a = 10
```

Explanation

The method() is overloaded in class Base whereas it is derived in class Derived with the double type as the parameter. In the method call, the integer is passed.

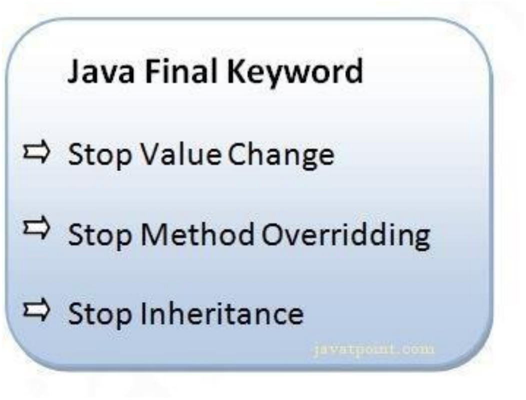
87) Can you have virtual functions in Java?

Yes, all functions in Java are virtual by default.

88) What is the final variable?

In Java, the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its value. In other words, we can say that the final variable

once assigned to a value, can never be changed after that. The final variable which is not assigned to any value can only be assigned through the class constructor.



```java
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
 speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```
Test it Now

```
Output:Compile Time Error
```

89) What is the final method?

If we change any method to a final method, we can't override it. More Details.

```java
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Test it Now

```
Output:Compile Time Error
```

90) What is the final class?

If we make any class final, we can't inherit it into any of the subclasses.

```java
final class Bike{}

 class Honda1 extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda1 honda= new Honda1();
   honda.run();
   }
}
```

Test it Now

```
Output:Compile Time Error
```

More Details.

91) What is the final blank variable?

A final variable, not initialized at the time of declaration, is known as the final blank variable. We can't initialize the final blank variable directly. Instead, we have to initialize it by using the class constructor. It is useful in the case when the user has some data which must not be changed by others, for example, PAN Number. Consider the following example:

```java
 class Student{
 int id;
 String name;
 final String PAN_CARD_NUMBER;
...
 }
```

92) Can we initialize the final blank variable?

Yes, if it is not static, we can initialize it in the constructor. If it is static blank final variable, it can be initialized only in the static block.

93) Can you declare the main method as final?

Yes, We can declare the main method as public static final void main(String[] args){}.

94) What is the output of the following Java program?

```java
class Main {
 public static void main(String args[]){
   final int i;
   i = 20;
   System.out.println(i);
 }
}
```

Output

```
20
```

Explanation

Since i is the blank final variable. It can be initialized only once. We have initialized it to 20. Therefore, 20 will be printed.

---

95) What is the output of the following Java program?

```java
class Base
{
    protected final void getInfo()
    {
        System.out.println("method of Base class");
    }
}

public class Derived extends Base
{
    protected final void getInfo()
    {
        System.out.println("method of Derived class");
    }
    public static void main(String[] args)
    {
        Base obj = new Base();
        obj.getInfo();
    }
}
```

Output

```
        Derived.java:11: error: getInfo() in Derived cannot override getInfo()
in Base
    protected final void getInfo()
                          ^
  overridden method is final
1 error
```

Explanation

The getDetails() method is final; therefore it can not be overridden in the subclass.

---

96) Can we declare a constructor as final?

The constructor can never be declared as final because it is never inherited. Constructors are not ordinary methods; therefore, there is no sense to declare constructors as final. However, if you try to do so, The compiler will throw an error.

---

97) Can we declare an interface as final?

No, we cannot declare an interface as final because the interface must be implemented by some class to provide its definition. Therefore, there is no sense to make an interface final. However, if you try to do so, the compiler will show an error.

---

98) What is the difference between the final method and abstract method?

The main difference between the final method and abstract method is that the abstract method cannot be final as we need to override them in the subclass to give its definition.

Core Java - OOPs: Polymorphism Interview Questions

99) What is the difference between compile-time polymorphism and runtime polymorphism?

There are the following differences between compile-time polymorphism and runtime polymorphism.

SN	compile-time polymorphism	Runtime polymorphism
1	In compile-time polymorphism, call to a method is resolved at compile-time.	In runtime polymorphism, call to an overridden method is resolved at runtime.

2	It is also known as static binding, early binding, or overloading.	It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch.
3	Overloading is a way to achieve compile-time polymorphism in which, we can define multiple methods or constructors with different signatures.	Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class.
4	It provides fast execution because the type of an object is determined at compile-time.	It provides slower execution as compare to compile-time because the type of an object is determined at run-time.
5	Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time.	Run-time polymorphism provides more flexibility because all the things are resolved at runtime.

100) What is Runtime Polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

```java
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}
  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Test it Now

Output:

```
running safely with 60km.
```

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

More details.

---

101) Can you achieve Runtime Polymorphism by data members?

No, because method overriding is used to achieve runtime polymorphism and data members cannot be overridden. We can override the member functions but not the data members. Consider the example given below.

```java
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;
 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj.speedlimit);//90
  }
```

Test it Now

Output:

```
90
```

102) What is the difference between static binding and dynamic binding?

In case of the static binding, the type of the object is determined at compile-time whereas, in the dynamic binding, the type of the object is determined at runtime.

Static Binding

```java
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

Dynamic Binding

```java
class Animal{
 void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
 void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

More details.

---

103) What is the output of the following Java program?

```java
class BaseTest
{
  void print()
  {
    System.out.println("BaseTest:print() called");
  }
}
public class Test extends BaseTest
{
  void print()
  {
    System.out.println("Test:print() called");
  }
  public static void main (String args[])
  {
    BaseTest b = new Test();
    b.print();
  }
}
```

Output

```
Test:print() called
```

Explanation

It is an example of Dynamic method dispatch. The type of reference variable b is determined at runtime. At compile-time, it is checked whether that method is present in the

Base class. In this case, it is overridden in the child class, therefore, at runtime the derived class method is called.

---

104) What is Java instanceOf operator?

The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has a null value, it returns false. Consider the following example.

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

Test it Now

Output

```
true
```

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

Core Java - OOPs Concepts: Abstraction Interview Questions

105) What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it. Abstraction lets you focus on what the object does instead of how it does it.

In Java, there are two ways to achieve the abstraction.

- Abstract Class
- Interface

More details.

---

106) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

107) What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method. Consider the following example.

```java
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Test it Now

Output

```
running safely
```

More details.

---

108) Can there be an abstract method without an abstract class?

No, if there is an abstract method in a class, that class must be abstract.

---

109) Is the following program written correctly? If yes then what will be the output of the program?

```java
abstract class Calculate
{
    abstract int multiply(int a, int b);
}

public class Main
{
    public static void main(String[] args)
    {
        int result = new Calculate()
        {
            @Override
            int multiply(int a, int b)
            {
                return a*b;
            }
        }.multiply(12,32);
        System.out.println("result = "+result);
    }
}
```

Yes, the program is written correctly. The Main class provides the definition of abstract method multiply declared in abstract class Calculation. The output of the program will be:

Output

```
384
```

110) Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

111) Is it possible to instantiate the abstract class?

No, the abstract class can never be instantiated even if it contains a constructor and all of its methods are implemented.

112) What is the interface?

The interface is a blueprint for a class that has static constants and abstract methods. It can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

113) Can you declare an interface method static?

No, because methods of an interface are abstract by default, and we can not use static and abstract together.

114) Can the Interface be final?

No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

---

115) What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

```
public interface Serializable{
}
```

---

116) What are the differences between abstract class and interface?

Abstract class	Interface
An abstract class can have a method body (non-abstract methods).	The interface has only abstract methods.
An abstract class can have instance variables.	An interface cannot have instance variables.
An abstract class can have the constructor.	The interface cannot have the constructor.
An abstract class can have static methods.	The interface cannot have static methods.
You can extend one abstract class.	You can implement multiple interfaces.
The abstract class **can provide the implementation of the interface**.	The Interface **can't provide the implementation of the abstract class**.
The **abstract keyword** is used to declare an abstract class.	The **interface keyword** is used to declare an interface.
An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.

An **abstract class** can be extended using keyword **extends**	An **interface class** can be implemented using keyword **implements**
A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

117) Can we define private and protected modifiers for the members in interfaces?

No, they are implicitly public.

118) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

119) How to make a read-only class in Java?

A class can be made read-only by making all of the fields private. The read-only class will have only getter methods which return the private property of the class to the main method. We cannot modify this property because there is no setter method available in the class. Consider the following example.

```
 //A Java class which has only getter methods.
public class Student{
//private data member
private String college="AKG";
//getter method for college
public String getCollege(){
return college;
}
 }
```

120) How to make a write-only class in Java?

A class can be made write-only by making all of the fields private. The write-only class will have only setter methods which set the value passed from the main method to the private fields. We cannot read the properties of the class because there is no getter method in this class. Consider the following example.

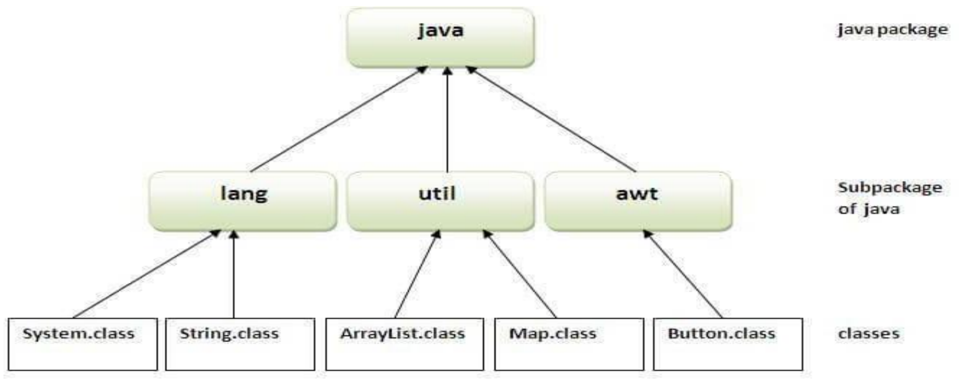
```
  //A Java class which has only setter methods.
public class Student{
//private data member
private String college;
//getter method for college
public void setCollege(String college){
this.college=college;
}
 }
```

Core Java - OOPs Concepts: Package Interview Questions

121) What is the package?

A package is a group of similar type of classes, interfaces, and sub-packages. It provides access protection and removes naming collision. The packages in Java can be categorized into two forms, inbuilt package, and user-defined package. There are many built-in packages such as Java, lang, awt, javax, swing, net, io, util, sql, etc. Consider the following example to create a package in Java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
   }
}
```



122) What are the advantages of defining packages in Java?

By defining packages, we can avoid the name conflicts between the same class names defined in different packages. Packages also enable the developer to organize the similar classes more effectively. For example, one can clearly understand that the classes present in java.io package are used to perform io related operations.

---

123) Do I need to import java.lang package any time? Why?

No. It is by default loaded internally by the JVM.

---

124) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or the same class multiple times. Neither compiler nor JVM complains about it. However, the JVM will internally load the class only once no matter how many times you import the same class.

---

125) What is the static import?

By static import, we can access the static members of a class directly, and there is no to qualify it with the class name.

More details.

Java: Exception Handling Interview Questions

126) How many types of exception can occur in a Java program?

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

- **Checked Exception:** Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.
- **Unchecked Exception:** Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- **Error:** Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

---

127) What is Exception Handling?

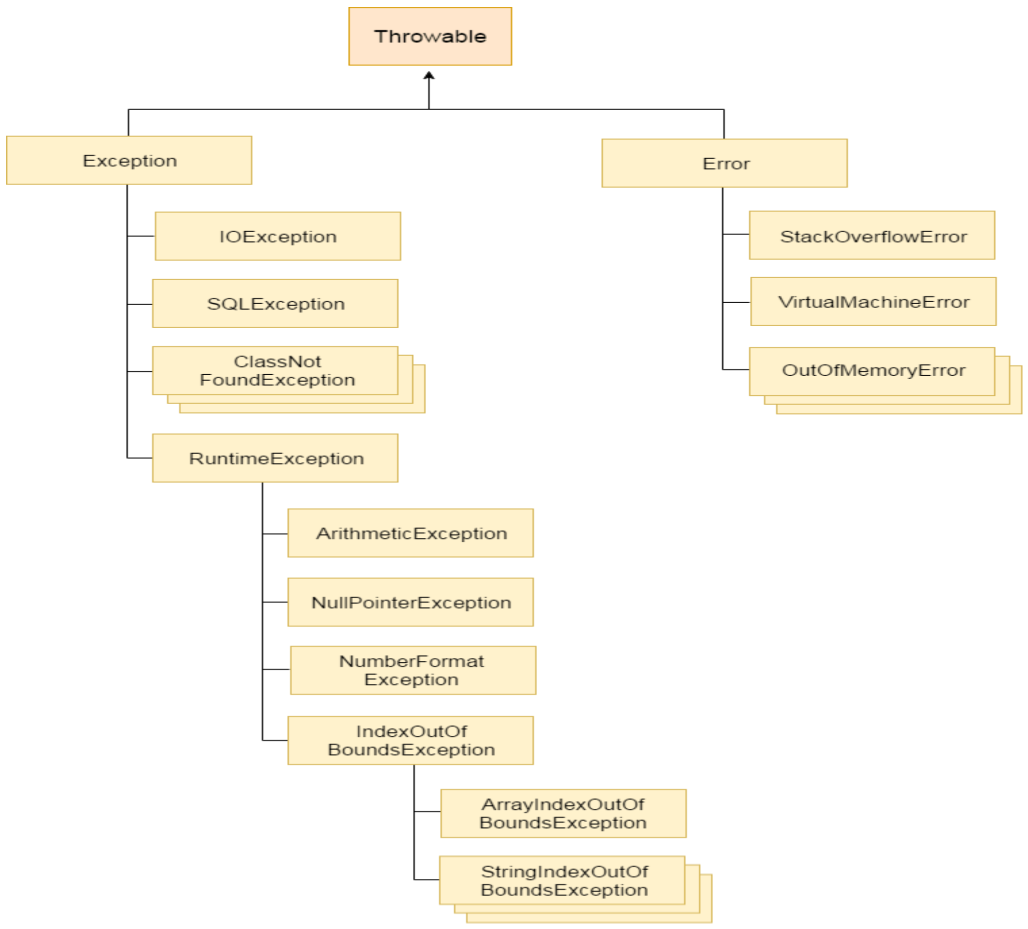
Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program. There are mainly two types of exceptions: checked and unchecked. Here, the error is considered as the unchecked exception.

More details.

---

128) Explain the hierarchy of Java Exception classes?

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



---

129) What is the difference between Checked Exception and Unchecked Exception?

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions, e.g., IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions, e.g., ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time.

135) What is the base class for Error and Exception?

The Throwable class is the base class for Error and Exception.

136) Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. So whatever exceptions are likely to be thrown should be declared in the throws clause of the method. Consider the following example.

```
public class Main{
    public static void main(String []args){
      try{
         int a = 1;
      System.out.println(a/0);
      }
      finally
      {
         System.out.println("rest of the code...");
      }
    }
}

```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

137) What is the output of the following Java program?

```
public class ExceptionHandlingExample {
public static void main(String args[])
{
   try
```

```
    {
        int a = 1/0;
        System.out.println("a = "+a);
    }
    catch(Exception e){System.out.println(e);}
    catch(ArithmeticException ex){System.out.println(ex);}
}
}
```

Output

```
ExceptionHandlingExample.java:10: error: exception ArithmeticException has already been caught
        catch(ArithmeticException ex){System.out.println(ex);}
        ^
1 error
```

Explanation

ArithmaticException is the subclass of Exception. Therefore, it can not be used after Exception. Since Exception is the base class for all the exceptions, therefore, it must be used at last to handle the exception. No class can be used after this.

---

138) What is finally block?

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection. Here, we must know that for each try block there can be zero or more catch blocks, but only one finally block. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

---

139) Can finally block be used without a catch?

Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch.

---

140) Is there any case when finally will not be executed?

Finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

---

141) What is the difference between throw and throws?

throw keyword	throws keyword
1) The **throw** keyword is used to throw an exception explicitly.	The **throws** keyword is used to declare an exception.
2) The checked exceptions cannot be propagated with throw only.	The checked exception can be propagated with throws
3) The **throw** keyword is followed by an instance.	The **throws** keyword is followed by class.
4) The **throw** keyword is used within the method.	The **throws** keyword is used with the method signature.
5) You cannot throw multiple exceptions.	You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

More details.

---

142) What is the output of the following Java program?

```java
public class Main{
    public static void main(String []args){
        try
        {
            throw 90;
        }
        catch(int e){
        System.out.println("Caught the exception "+e);
        }

    }
}
```

Output

```
Main.java:6: error: incompatible types: int cannot be converted to Throwable
            throw 90;
            ^
Main.java:8: error: unexpected type
        catch(int e){
              ^
  required: class
  found:    int
2 errors
```

Explanation

In Java, the throwable objects can only be thrown. If we try to throw an integer object, The compiler will show an error since we can not throw basic data type from a block of code.

---

143) What is the output of the following Java program?

```java
class Calculation extends Exception
{
    public Calculation()
    {
        System.out.println("Calculation class is instantiated");
    }
    public void add(int a, int b)
    {
        System.out.println("The sum is "+(a+b));
    }
}
public class Main{
     public static void main(String []args){
        try
        {
            throw new Calculation();
        }
        catch(Calculation c){
            c.add(10,20);
        }
    }
}
```

Output

```
Calculation class is instantiated
The sum is 30
```

Explanation

The object of Calculation is thrown from the try block which is caught in the catch block. The add() of Calculation class is called with the integer values 10 and 20 by using the object of this class. Therefore there sum 30 is printed. The object of the Main class can only be thrown in the case when the type of the object is throwable. To do so, we need to extend the throwable class.

---

144) Can an exception be rethrown?

Yes.

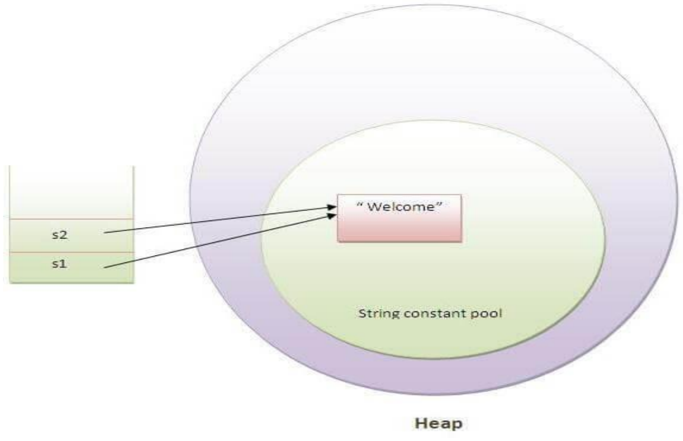
145) Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Yes but only unchecked exception not checked.

Java: String Handling Interview Questions

146) What is String Pool?

String pool is the space reserved in the heap memory that can be used to store the strings. The main advantage of using the String pool is whenever we create a string literal; the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. Therefore, it saves the memory by avoiding the duplicacy.



147) What is the meaning of immutable regarding String?

The simple meaning of immutable is unmodifiable or unchangeable. In Java, String is immutable, i.e., once string object has been created, its value can't be changed. Consider the following example for better understanding.

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
 System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

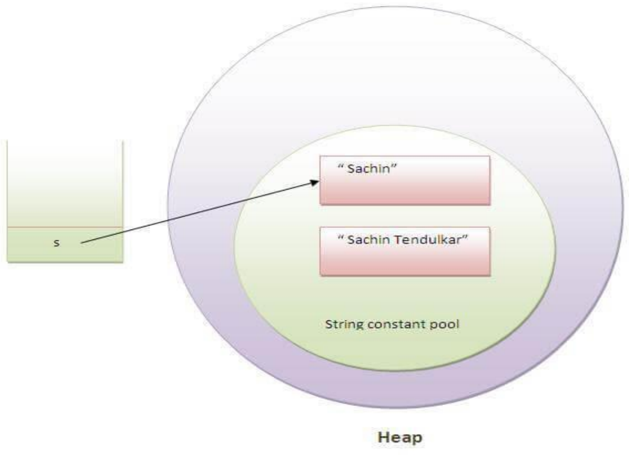
Test it Now

Output:

```
Sachin
```

148) Why are the objects immutable in java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.



149) How many ways can we create the string object?

1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. String objects are stored in a special memory area known as the **string constant pool** For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance
```

2) By new keyword

```
1. String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the constant string pool. The variable s will refer to the object in a heap (non-pool).

153) How many objects will be created in the following code?

```java
String s1="Welcome";
String s2="Welcome";
String s3="Welcome";
```

Only one object will be created using the above code because strings in Java are immutable.

---

154) Why java uses the concept of the string literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

---

155) How many objects will be created in the following code?

```java
String s = new String("Welcome");
```

Two objects, one in string constant pool and other in non-pool(heap).

---

156) What is the output of the following Java program?

```java
public class Test

  public static void main (String args[])
  {
      String a = new String("Sharma is a good player");
      String b = "Sharma is a good player";
      if(a == b)
      {
          System.out.println("a == b");
      }
      if(a.equals(b))
      {
          System.out.println("a equals b");
      }
  }
```

Output

```
a equals b
```

Explanation

The operator **==** also check whether the references of the two string objects are equal or not. Although both of the strings contain the same content, their references are not equal because both are created by different ways(Constructor and String literal) therefore, **a ==**

b is unequal. On the other hand, the equal() method always check for the content. Since their content is equal hence, **a equals b** is printed.

157) What is the output of the following Java program?

```
public class Test
{
    public static void main (String args[])
    {
        String s1 = "Sharma is a good player";
        String s2 = new String("Sharma is a good player");
        s2 = s2.intern();
    System.out.println(s1 ==s2);
    }
}
```

Output

```
true
```

Explanation

The intern method returns the String object reference from the string pool. In this case, s1 is created by using string literal whereas, s2 is created by using the String pool. However, s2 is changed to the reference of s1, and the operator **==** returns true.

158) What are the differences between String and StringBuffer?

The differences between the String and StringBuffer is given in the table below.

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	The String is slow and consumes more memory when you concat too many strings because every time it creates a new instance.	The StringBuffer is fast and consumes less memory when youcancat strings.
3)	The String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	The StringBuffer class doesn't override the equals() method of Object class.

159) How can we create an immutable class in Java?

We can create an immutable class by defining a final class having all of its members as final. Consider the following example.

```
public final class Employee{
final String pancardNumber;

public Employee(String pancardNumber){
this.pancardNumber=pancardNumber;
}

public String getPancardNumber(){
return pancardNumber;
}

}
```

160) What is the purpose of toString() method in Java?

The toString() method returns the string representation of an object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object, etc. depending upon your implementation. By overriding the toString() method of the Object class, we can return the values of the object, so we don't need to write much code. Consider the following example.

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public String toString(){//overriding the toString() method
  return rollno+" "+name+" "+city;
 }
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");
 
  System.out.println(s1);//compiler writes here s1.toString()
  System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

Output:

```
101 Raj lucknow
102 Vijay ghaziabad
```

More details.

---

161) Why CharArray() is preferred over String to store the password?

String stays in the string pool until the garbage is collected. If we store the password into a string, it stays in the memory for a longer period, and anyone having the memory-dump can extract the password as clear text. On the other hand, Using CharArray allows us to set it to blank whenever we are done with the password. It avoids the security threat with the string by enabling us to control the memory.

---

162) Write a Java program to count the number of words present in a string?

Program:

```
  public class Test
{
    public static void main (String args[])
    {
        String s = "Sharma is a good player and he is so punctual";
        String words[] = s.split(" ");
        System.out.println("The Number of words present in the string are : "+words.length);

    }
}
```

Output

```
The Number of words present in the string are : 10
```

---

163) Write a regular expression to validate a password. A password must start with an alphabet and followed by

alphanumeric characters; Its length must be in between 8 to 20.

The regular expression for the above criteria will be: **^[a-zA-Z][a-zA-Z0-9]{8,19}** where ^ represents the start of the regex, [a-zA-Z] represents that the first character must be an alphabet, [a-zA-Z0-9] represents the alphanumeric character, {8,19} represents that the length of the password must be in between 8 and 20.

---

164) What is the output of the following Java program?

```
1. import java.util.regex.*;
2. class RegexExample2{
3. public static void main(String args[]){
4. System.out.println(Pattern.matches(".s", "as")); //line 4
5. System.out.println(Pattern.matches(".s", "mk")); //line 5
6. System.out.println(Pattern.matches(".s", "mst")); //line 6
7. System.out.println(Pattern.matches(".s", "amms")); //line 7
8. System.out.println(Pattern.matches("..s", "mas")); //line 8
9. }}
```

Output

```
true
false
false
false
true
```

Explanation

line 4 prints true since the second character of string is s, line 5 prints false since the second character is not s, line 6 prints false since there are more than 3 characters in the string, line 7 prints false since there are more than 2 characters in the string, and it contains more than 2 characters as well, line 8 prints true since the third character of the string is s.

Core Java: Nested classes and Interfaces Interview Questions

165) What are the advantages of Java inner classes?

There are two types of advantages of Java inner classes.

- Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of the outer class including private.
- Nested classes are used to develop a more readable and maintainable code because it logically groups classes and interfaces in one place only.
- **Code Optimization:** It requires less code to write.

---

166) What is a nested class?

The nested class can be defined as the class which is defined inside another class or interface. We use the nested class to logically group classes and interfaces in one place so that it can be more readable and maintainable. A nested class can access all the data members of the outer class including private data members and methods. The syntax of the nested class is defined below.

```
class Java_Outer_class{
 //code
 class Java_Nested_class{
  //code
 }
}

```

There are two types of nested classes, static nested class, and non-static nested class. The non-static nested class can also be called as inner-class

167) What are the types of inner classes (non-static nested class) used in Java?

There are mainly three types of inner classes used in Java.

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within the method.

168) Is there any difference between nested classes and inner classes?

Yes, inner classes are non-static nested classes. In other words, we can say that inner classes are the part of nested classes.

169) Can we access the non-final local variable, inside the local inner class?

No, the local variable must be constant if you want to access it in the local inner class.

170) What are anonymous inner classes?

Anonymous inner classes are the classes that are automatically declared and instantiated within an expression. We cannot apply different access modifiers to them

171) What is the nested interface?

An Interface that is declared inside the interface or class is known as the nested interface.

```
interface interface_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

172) Can a class have an interface?

Yes, an interface can be defined within the class. It is called a nested interface.

173) Can an Interface have a class?

Yes, they are static implicitly.

Garbage Collection Interview Questions

174) What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management.

175) What is gc()?

The gc() method is used to invoke the garbage collector for cleanup processing. This method is found in System and Runtime classes. This function explicitly makes the Java Virtual Machine free up the space occupied by the unused objects so that it can be utilized or reused. Consider the following example for the better understanding of how the gc() method invoke the garbage collector.

```java
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
 System.gc();
 }
}
```

Test it Now

```
object is garbage collected
object is garbage collected
```

176) How is garbage collection controlled?

Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

177) How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

1) By nulling a reference:

```java
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```java
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```java
new Employee();
```

---

183) What is the purpose of the finalize() method?

The finalize() method is invoked just before the object is garbage collected. It is used to perform cleanup processing. The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created an object without new, you can use the finalize method to perform cleanup processing (destroying remaining objects). The cleanup processing is the process to free up all the resources, network which was previously used and no longer needed. It is essential to remember that it is not a reserved keyword, finalize method is present in the object class hence it is available in every class as object class is the superclass of every class in java. Here, we must note that neither finalization nor garbage collection is guaranteed. Consider the following example.

```java
public class FinalizeTest {
    int j=12;
    void add()
    {
        j=j+12;
        System.out.println("J="+j);
    }
    public void finalize()
    {
        System.out.println("Object is garbage collected");
    }
    public static void main(String[] args) {
        new FinalizeTest().add();
        System.gc();
        new FinalizeTest().add();
    }
}

```

---

184) Can an unreferenced object be referenced again?

Yes

---

185) What kind of thread is the Garbage collector thread?

Daemon thread.

---

186) What is the difference between final, finally and finalize?

No.	final	Finally	finalize
1)	Final is used to apply restrictions on class, method, and variable. The final class can't be inherited, final method can't be overridden, and final variable value can't be changed.	Finally is used to place important code, it will be executed whether an exception is handled or not.	Finalize is used to perform clean up processing just before an object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

187) What is the purpose of the Runtime class?

Java Runtime class is used to interact with a java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory, etc. There is only one instance of java.lang.Runtime class is available for one java application. The Runtime.getRuntime() method returns the singleton instance of Runtime class.

188) How will you invoke any external process in Java?

By Runtime.getRuntime().exec(?) method. Consider the following example.

```
public class Runtime1{
 public static void main(String args[])throws Exception{
 Runtime.getRuntime().exec("notepad");//will open a new notepad
 }
}
```

I/O Interview Questions

189) What do you understand by an IO stream?

The stream is a sequence of data that flows from source to destination. It is composed of bytes. In Java, three streams are created for us automatically.

- System.out: standard output stream
- System.in: standard input stream
- System.err: standard error stream

---

190) What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented. The ByteStream classes are used to perform input-output of 8-bit bytes whereas the CharacterStream classes are used to perform the input/output for the 16-bit Unicode system. There are many classes in the ByteStream class hierarchy, but the most frequently used classes are FileInputStream and FileOutputStream. The most frequently used classes CharacterStream class hierarchy is FileReader and FileWriter.

---

191) What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

Java BufferedOutputStream class is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. Whereas, Java BufferedInputStream class is used to read information from the stream. It internally uses the buffer mechanism to make the performance fast.

---

192) In Java, How many ways you can take input from the console?

In Java, there are three ways by using which, we can take input from the console.

- **Using BufferedReader class:** we can take input from the console by wrapping System.in into an InputStreamReader and passing it into the BufferedReader. It provides an efficient reading as the input gets buffered. Consider the following example.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Person
{
    public static void main(String[] args) throws IOException
    {
      System.out.println("Enter the name of the person");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String name = reader.readLine();
        System.out.println(name);
    }
}
```

- **Using Scanner class:** The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using a regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces. Consider the following example.

```java
import java.util.*;
public class ScannerClassExample2 {
      public static void main(String args[]){
         String str = "Hello/This is JavaTpoint/My name is Abhishek.";
         //Create scanner with the specified String Object
         Scanner scanner = new Scanner(str);
         System.out.println("Boolean Result: "+scanner.hasNextBoolean());
         //Change the delimiter of this scanner
         scanner.useDelimiter("/");
         //Printing the tokenized Strings
         System.out.println("---Tokenizes String---");
       while(scanner.hasNext()){
           System.out.println(scanner.next());
       }
         //Display the new delimiter
         System.out.println("Delimiter used: " +scanner.delimiter());
         scanner.close();
         }
}

```

- **Using Console class:** The Java Console class is used to get input from the console. It provides methods to read texts and passwords. If you read the password using the Console class, it will not be displayed to the user. The java.io.Console class is attached to the system console internally. The Console class is introduced since 1.5. Consider the following example.

```java
import java.io.Console;
class ReadStringTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome "+n);
}
 }
```